Aluno: Jonatan F. Hartmann - 01208A

pyFood - Sistema de pedidos entre lojas e clientes

1. RESUMO DAS FUNCIONALIDADES

O trabalho final consiste em um sistema de pedidos de delivery entre lojas e clientes (semelhante ao Ifood). Dessa maneira um usuário pode se cadastrar como loja, ou como cliente. A seguir, respectivas funcionalidades:

a. Loja:
    1) Cadastrar, atualizar, buscar e deletar diferentes produtos;
    2) Gerenciar o histórico de pedidos, quantidade de vendas e o total obtido.
b. Cliente:
    1) Realizar pedidos buscando produtos nas lojas cadastradas;
    2) Gerenciar o histórico de pedidos já feitos.

Como observação, a superclasse "Usuário" define o método “logar(login, senha)” que - por polimorfismo - acessa o menu de ações da respectiva loja ou cliente.

Além do mais, definiu-se os Produtos contendo uma lista de “Adicionais” que por sua vez são adições possíveis com respectivos valores para determinado produto cadastrado.

Outrossim, ao realizar um pedido, o cliente pode utilizar algum cupom da sua “lista de cupons”, este antes mesmo de ser utilizado pelo cliente, para ser criado deve ser validado (de acordo com os cupons que serão definidos pelo código fonte) determinando assim a quantidade de desconto, a categoria de loja que se aplica e o valor mínimo do pedido para ser utilizado. Seu uso é único por pedido, então assim que utilizado, é removido automaticamente da lista de cupons do cliente. O cliente também pode adicionar mais cupons em sua lista (desde que seu código seja válido de acordo com o código fonte) através do método add_cupom(cupom). Códigos possíveis de cupom:

a) “DESC10”: 10% de desconto, qualquer categoria, compra mínima 15 R$;
b) “DESC15”: 15% de desconto, qualquer categoria, compra mínima 30 R$;
c) “RESTAURANTOFF”: 20% de desconto, restaurantes, compra mínima 20 R$;
d) “SWEETPRICE”: 5% de desconto, sorveterias, compra mínima 0 R$.

Por fim, cada pedido, produto e adicional possui um código associado gerado no momento em que seu respectivo objeto for instanciado. Abaixo é definido como cada código é gerado:

a) pedido: “login da loja”+”ddMMAA”+”3 últimos dígitos do CPF do cliente”+”HHmmss”;
b) produto: “índice de produto na lista de produtos”;
c) adicional: ”índice do adicional na lista de adicionais”.

2. INTERFACE

O sistema foi organizado de modo que a sua funcionalidade (classes principais e seus métodos) sejam independentes da interface utilizada. Dessa forma seria possível fazer interfaces diferentes e tornar o sistema útil apenas chamando os métodos adequadamente de cada classe.

Entretanto, para melhor uso do sistema, optei por utilizar a biblioteca Eel para desenvolver a interface. Esta permite que suas funções/métodos em python sejam utilizados no javascript, possibilitando assim construir a interface na web. Desta maneira, o pyFood possui suas classes principais funcionais por si só, e uma classe Interface, com

métodos mapeados para o javascript (com o uso da Eel), para tornar o sistema utilizável na web.

Assim, a interface web foi desenvolvida com o uso do bootstrap para estilização, e a interação do usuário com o jQuery utilizado no javascript. Assim sendo, qualquer interação do usuário que envolve alguma lógica do backend desenvolvido com python, é executada chamando primeiro a função javascript referida, e esta chama a função interna desenvolvida na classe Interface de python.

Desse modo, para iniciar o sistema, basta executar o arquivo Main.py, que possui apenas a instanciação da classe Interface, visto que o próprio construtor da interface faz o processo de iniciar a biblioteca Eel mapeando os métodos do python para o javascript.

3. PERSISTÊNCIA DOS DADOS

Por fim, todos os dados envolvidos na aplicação são salvos em arquivos .json (pois permitem um melhor gerenciamento como dicionários quando acessados como python) através da classe DataFiles criada especialmente para isso.

Os arquivos json utilizados e suas utilidades são:

1) usuarios.json: armazena todos os usuários (lojas e clientes) do sistema. Cada usuário contém: login, senha e tipo de usuário.
2) clientes.json: armazena todos os clientes do sistema, cada um desses contendo: login, senha, nome, cpf, lista de cupons e um dicionário com histórico de pedidos (chave = id do pedido).
3) cuponsValidos.json: armazena os cupons válidos para serem obtidos pelos usuários através de uma chave que é seu próprio código de uso. Cada cupom possui: código de uso, categoria aplicável, valor mínimo do pedido a ser aplicado, e a porcentagem de desconto.
4) lojas.json: armazena todas as lojas do sistema, cada uma contendo: nome, login, senha, endereço, categoria, lista de produtos, quantidade de vendas, total das vendas, e um dicionário de histórico de vendas (chave = id do pedido associado).
5) pedidos.json: armazena todos os pedidos (vendas) feitos no sistema. Cada pedido é armazenado com uma chave específica (que é o código do pedido), contendo: código, dados da loja, dados do cliente, endereço de entrega, lista de produtos comprados, data/hora realizado, taxa de entrega, total, subtotal e cupons utilizados.

4. OBSERVAÇÕES FINAIS

Vale lembrar que a biblioteca Eel deve ser instalada caso não a possua no seu sistema (o python vai lançar um erro de biblioteca a ser importada não encontrada). Para isso ser feito pode ser utilizado o instalador de pacotes do python *pip*, de qualquer modo mais informações sobre podem ser encontradas na documentação da biblioteca.

Além disso, para desenvolver a interface eu utilizei algumas bibliotecas de estilo (e o próprio jQuery) importando-as da web sem tê-las instaladas de fato no sistema. Nesse sentido, ao executar o sistema possivelmente offline, algumas funcionalidades serão quebradas (por conta da interface). As bibliotecas utilizadas foram: jQuery, Bootstrap, SweetAlert e Select2.